Laporan Final Project Mikrokontroller

Disusun guna memenuhi Evaluasi Akhir Semester mata kuliah Mikrokontroller

Dosen Pengampu:

Dr. Basuki Rahmat, S.Si. MT

Disusun oleh:

Miftahus Syifaul Haqqi

(23081010197)

PROGRAM STUDI INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN" JAWA TIMUR

TAHUN 2025

Percobaan 1 (Menghidupkan LED Sederhana)

Kode Program

- Arduino

```
int ledPin = 2;
char myData = 0;

int motor1Pin1 = 27;
int motor1Pin2 = 26;
int enable1Pin = 12;

// PWM ESP32
const int freq = 30000;
const int pwmChannel = 0;
const int resolution = 8; // 0..255
int dutyCycle = 0;

const byte pin_rpm = 13;
volatile int rev = 0; // belum dipakai

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);

  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);

  pinMode(pin_rpm, INPUT_PULLUP);
  // attachInterrupt(digitalPinToInterrupt(pin_rpm), isr, RISING);

  ledcSetup(pwmChannel, freq, resolution);
  ledcAttachPin(enable1Pin, pwmChannel);

  // Pastikan motor OFF saat awal
  MotorOff();

  Serial.println("Siap. Kirim '1' untuk ON, '0' untuk OFF.");
}

void MotorOn() {
  dutyCycle = 205; // ≈80%
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
```

```
  ledcWrite(pwmChannel, dutyCycle);
}

void MotorOff() {
  dutyCycle = 0;
  // COAST: lepas kedua sisi; ganti ke HIGH/HIGH untuk BRAKE
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  ledcWrite(pwmChannel, dutyCycle);
}

void loop() {
  // Baca hanya jika ada data
  if (Serial.available() > 0) {
    char c = Serial.read();

    // Abaikan newline / carriage return
    if (c == '\n' || c == '\r') return;

    if (c == '1') {
      digitalWrite(ledPin, HIGH);
      MotorOn();
      Serial.println("LED ON, Motor ON (≈80%)");
    } else if (c == '0') {
      digitalWrite(ledPin, LOW);
      MotorOff();
      Serial.println("LED OFF, Motor OFF");
    } else {
      Serial.print("Perintah tidak dikenal: ");
      Serial.println(c);
      Serial.println("Kirim '1' atau '0'.");
    }
  }

}
```

Penjelasan Kode Program

Pada bagian awal program dideklarasikan beberapa variabel yang digunakan untuk mengatur LED dan motor DC. Variabel ledPin digunakan untuk menentukan pin GPIO ESP32 yang terhubung dengan LED. Variabel motor1Pin1 dan motor1Pin2 digunakan untuk mengatur arah putaran motor, sedangkan enable1Pin digunakan sebagai pin PWM untuk mengatur daya motor. Variabel myData digunakan untuk menyimpan data yang diterima dari Python melalui komunikasi serial.

Selanjutnya, program mendefinisikan parameter PWM yang terdiri dari frekuensi, channel PWM, resolusi PWM, dan nilai dutyCycle. Frekuensi PWM diatur sebesar 30 kHz untuk menghasilkan sinyal yang stabil pada motor. Resolusi PWM sebesar 8 bit digunakan sehingga nilai PWM berada pada rentang 0 hingga 255. Variabel dutyCycle berfungsi sebagai penentu kecepatan motor.

Pada fungsi setup(), pin LED dan seluruh pin motor dikonfigurasikan sebagai output. Komunikasi serial diinisialisasi dengan baud rate 115200 agar ESP32 dapat menerima perintah dari Python. Selanjutnya, konfigurasi PWM dilakukan menggunakan fungsi ledcSetup() dan pin enable1Pin dihubungkan ke channel PWM menggunakan fungsi ledcAttachPin(). Pada akhir fungsi setup(), fungsi MotorOff() dipanggil untuk memastikan motor dalam kondisi mati saat sistem pertama kali dijalankan.

Fungsi MotorOn() digunakan untuk menyalakan motor DC. Pada fungsi ini, nilai dutyCycle diatur sebesar 205 sehingga motor berputar dengan kecepatan tertentu. Pin motor1Pin1 diberi logika HIGH dan motor1Pin2 diberi logika LOW sehingga motor berputar ke satu arah. Nilai PWM kemudian dikirim ke motor menggunakan fungsi ledcWrite().

Fungsi MotorOff() digunakan untuk mematikan motor DC. Pada fungsi ini, nilai dutyCycle diatur menjadi 0 dan kedua pin arah motor diberi logika LOW sehingga motor berhenti berputar.

Pada fungsi loop(), program secara terus-menerus memeriksa apakah terdapat data yang masuk melalui komunikasi serial. Jika data tersedia, maka karakter yang diterima akan disimpan ke dalam variabel myData. Apabila karakter yang diterima adalah '1', maka LED akan dinyalakan dan fungsi MotorOn() dipanggil untuk menyalakan motor. Sebaliknya, apabila karakter yang diterima adalah '0', maka LED akan dimatikan dan fungsi MotorOff() dipanggil sehingga motor berhenti.

Dengan demikian, program Arduino pada Modul 1 berhasil mengimplementasikan pengendalian LED dan motor DC secara ON dan OFF berdasarkan perintah yang dikirimkan dari Python melalui komunikasi serial.

- Python

```python
import serial
#arduinoData = serial.Serial('/dev/ttyACM1', 9600)
arduinoData = serial.Serial('COM9', 115200, timeout=0)

while True:
    myData = input('Kirimkan perintah (1/0): ')
    if myData == "1":
        arduinoData.write(b'1')
    elif myData == "0":
       arduinoData.write(b'0')
```

Penjelasan Kode Program

Program Python pada Modul 1 digunakan sebagai pengirim perintah ke ESP32 melalui komunikasi serial. Pada awal program, library serial diimpor untuk memungkinkan komunikasi serial antara komputer dan ESP32. Setelah itu, objek serial dibuat dengan menentukan port COM yang digunakan, baud rate sebesar 115200, dan waktu tunggu koneksi.

Program kemudian mengirimkan karakter '1' ke ESP32 untuk menyalakan LED dan motor. Karakter tersebut dikirim dalam bentuk byte agar dapat diterima oleh ESP32 melalui komunikasi serial. Setelah beberapa saat, program mengirimkan karakter '0' untuk mematikan LED dan motor. Dengan cara ini, Python berfungsi sebagai pengendali utama, sedangkan ESP32 bertindak sebagai eksekutor perintah.

Percobaan 2 Menggerakkan Motor Menggunakan MQTT panel

Kode Program

- Arduino

```
int ledPin = 2;
char myData = 0;

int motor1Pin1 = 27;
int motor1Pin2 = 26;
int enable1Pin = 12;

// PWM ESP32
const int freq = 30000;
const int pwmChannel = 0;
const int resolution = 8; // 0..255
int dutyCycle = 0;

const byte pin_rpm = 13;
volatile int rev = 0; // belum dipakai

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);

  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);

  pinMode(pin_rpm, INPUT_PULLUP);
  // attachInterrupt(digitalPinToInterrupt(pin_rpm), isr, RISING);
```

```
  ledcSetup(pwmChannel, freq, resolution);
  ledcAttachPin(enable1Pin, pwmChannel);

  // Pastikan motor OFF saat awal
  MotorOff();

  Serial.println("Siap. Kirim '1' untuk ON, '0' untuk OFF.");
}

void MotorOn() {
  dutyCycle = 205; // ≈80%
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  ledcWrite(pwmChannel, dutyCycle);
}

void MotorOff() {
  dutyCycle = 0;
  // COAST: lepas kedua sisi; ganti ke HIGH/HIGH untuk BRAKE
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  ledcWrite(pwmChannel, dutyCycle);
}

void loop() {
  // Baca hanya jika ada data
  if (Serial.available() > 0) {
    char c = Serial.read();

    // Abaikan newline / carriage return
    if (c == '\n' || c == '\r') return;

    if (c == '1') {
      digitalWrite(ledPin, HIGH);
      MotorOn();
      Serial.println("LED ON, Motor ON (≈80%)");
    } else if (c == '0') {
      digitalWrite(ledPin, LOW);
      MotorOff();
      Serial.println("LED OFF, Motor OFF");
    } else {
      Serial.print("Perintah tidak dikenal: ");
      Serial.println(c);
      Serial.println("Kirim '1' atau '0'.");
```

```
    }
  }

}
```

Penjelasan Kode Program

Pada Modul 2, sistem dikembangkan untuk mengendalikan motor DC secara ON dan OFF menggunakan komunikasi nirkabel berbasis WiFi dan protokol MQTT. ESP32 berfungsi sebagai perangkat utama yang terhubung ke jaringan WiFi dan menerima perintah dari broker MQTT untuk mengendalikan motor dan LED indikator.

Pada bagian awal program, dilakukan pemanggilan library WiFi.h dan PubSubClient.h yang digunakan untuk menghubungkan ESP32 ke jaringan WiFi serta berkomunikasi dengan broker MQTT. Selanjutnya, didefinisikan parameter jaringan WiFi berupa SSID dan password, serta alamat broker MQTT yang digunakan sebagai perantara komunikasi data.

Program kemudian mendefinisikan pin-pin yang digunakan, meliputi pin LED sebagai indikator status, dua pin arah motor yang terhubung ke driver motor, dan satu pin enable motor yang digunakan sebagai keluaran PWM. Selain itu, parameter PWM juga ditentukan, yaitu channel PWM, frekuensi PWM, dan resolusi PWM sebesar 8 bit yang menghasilkan nilai PWM antara 0 hingga 255.

Pada fungsi connectWiFi(), ESP32 melakukan proses koneksi ke jaringan WiFi hingga status koneksi berhasil. Setelah terhubung, alamat IP ESP32 ditampilkan melalui serial monitor sebagai indikator keberhasilan koneksi jaringan.

Fungsi reconnectMQTT() digunakan untuk memastikan ESP32 selalu terhubung ke broker MQTT. Apabila koneksi terputus, ESP32 akan mencoba melakukan koneksi ulang dan melakukan subscribe ke topik MQTT yang telah ditentukan, sehingga ESP32 dapat menerima pesan kontrol dari publisher.

Pengendalian motor dilakukan melalui dua fungsi utama, yaitu motorON() dan motorOFF(). Pada fungsi motorON(), arah putaran motor diatur dengan memberikan logika HIGH dan LOW pada pin driver motor. Motor kemudian dijalankan secara bertahap menggunakan metode soft start dengan menaikkan nilai PWM secara perlahan untuk menghindari lonjakan arus. LED indikator dinyalakan sebagai tanda bahwa motor dalam kondisi aktif. Sebaliknya, pada fungsi motorOFF(), nilai PWM diturunkan secara bertahap hingga nol untuk menghentikan motor secara halus, kemudian seluruh pin motor dimatikan dan LED indikator dipadamkan.

Fungsi callback() digunakan untuk memproses pesan yang diterima dari broker MQTT. Pesan yang diterima diubah menjadi bentuk string dan dibandingkan dengan perintah yang telah ditentukan. Jika pesan yang diterima bernilai "1" atau "ON", maka motor akan dinyalakan. Sebaliknya, jika pesan bernilai "0" atau "OFF", maka motor akan dimatikan.

Pada fungsi setup(), seluruh pin dikonfigurasikan sebagai output, PWM diinisialisasi, serta koneksi WiFi dan MQTT disiapkan. Sementara itu, pada fungsi loop(), ESP32 secara terus-menerus memeriksa status koneksi MQTT dan menjalankan fungsi client.loop() untuk memastikan pesan MQTT dapat diterima dan diproses secara real-time.

Dengan demikian, Modul 2 menunjukkan bahwa ESP32 dapat digunakan untuk mengendalikan motor DC secara jarak jauh menggunakan jaringan WiFi dan protokol MQTT, dengan sistem kontrol yang sederhana namun efektif.

Percobaan 3 Menghidupkan LCD menggunakan Fade Control

Kode Program

- Arduino

```
// ===== LED FADE CONTROL VIA PYTHON (ESP32) =====

int ledPin = 2;

// PWM LED
const int ledChannel = 1;
const int ledFreq = 5000;
const int ledResolution = 8; // 0–255

bool runFade = false;   // status fade
int brightness = 255;
int stepFade = -5;      // arah fade (turun)

void setup() {
  Serial.begin(115200);

  ledcSetup(ledChannel, ledFreq, ledResolution);
  ledcAttachPin(ledPin, ledChannel);

  ledcWrite(ledChannel, 0); // LED mati awal
  Serial.println("Siap. Kirim '1' START fade, '0' STOP");
}

void loop() {
  // ===== BACA PERINTAH DARI PYTHON =====
  if (Serial.available()) {
    char c = Serial.read();

    if (c == '1') {
      runFade = true;
```

```
      Serial.println("Fade START");
    }
    else if (c == '0') {
      runFade = false;
      ledcWrite(ledChannel, 0);
      Serial.println("Fade STOP, LED OFF");
    }
  }

  // ===== PROSES FADE =====
  if (runFade) {
    ledcWrite(ledChannel, brightness);
    brightness += stepFade;

    if (brightness <= 0 || brightness >= 255) {
      stepFade = -stepFade; // balik arah
    }

    delay(40); // kecepatan fade
  }
}
```

Penjelasan Kode Program

Pada Modul 3, sistem dirancang untuk mengendalikan efek perubahan intensitas cahaya LED (fade) dari terang ke redup secara otomatis dengan kendali dari Python melalui komunikasi serial. ESP32 berfungsi sebagai pengendali utama yang menghasilkan sinyal PWM untuk mengatur tingkat kecerahan LED.

Pada awal program, didefinisikan pin LED yang digunakan serta parameter PWM, meliputi channel PWM, frekuensi PWM, dan resolusi PWM sebesar 8 bit. Resolusi ini menghasilkan rentang nilai PWM antara 0 hingga 255 yang merepresentasikan tingkat kecerahan LED dari kondisi mati hingga menyala penuh.

Variabel runFade digunakan sebagai penanda status apakah proses fade sedang berjalan atau tidak. Variabel brightness menyimpan nilai kecerahan LED saat ini, sedangkan variabel stepFade menentukan arah dan besar perubahan kecerahan LED. Nilai negatif pada stepFade menandakan bahwa proses fade dimulai dari kondisi terang menuju redup.

Pada fungsi setup(), komunikasi serial diinisialisasi untuk menerima perintah dari Python. Selanjutnya, konfigurasi PWM dilakukan dengan mengatur channel PWM dan menghubungkan pin LED ke channel tersebut. LED diatur dalam kondisi mati pada awal program untuk memastikan tidak terjadi penyalaan LED secara tidak sengaja.

Pada fungsi loop(), ESP32 membaca data yang dikirimkan dari Python melalui komunikasi serial. Ketika ESP32 menerima karakter '1', variabel runFade diaktifkan sehingga proses fade LED dimulai. Sebaliknya, ketika karakter '0' diterima, proses fade dihentikan dan LED dimatikan.

Selama variabel runFade bernilai aktif, ESP32 secara berkala mengirimkan nilai PWM ke LED menggunakan fungsi ledcWrite(). Nilai kecerahan LED diubah secara bertahap dengan menambahkan nilai stepFade. Apabila nilai kecerahan mencapai batas minimum atau maksimum, arah perubahan kecerahan akan dibalik sehingga tercipta efek fade naik dan turun secara kontinu. Penundaan waktu digunakan untuk mengatur kecepatan perubahan kecerahan LED agar terlihat halus.

Dengan demikian, Modul 3 menunjukkan bahwa ESP32 mampu menghasilkan efek fade LED secara otomatis menggunakan PWM, dengan kendali sederhana dari Python melalui komunikasi serial. Modul ini memperlihatkan integrasi antara perangkat lunak dan perangkat keras dalam menghasilkan sistem kendali yang responsif dan fleksibel.

- Python

```python
import serial
import time

# Ganti COM sesuai PC kamu
esp = serial.Serial('COM9', 115200)
time.sleep(2)

print("1 = START LED FADE")
print("0 = STOP LED FADE")
print("q = KELUAR")

while True:
    cmd = input("Perintah: ")

    if cmd == '1':
        esp.write(b'1')
        print("Fade dimulai")
    elif cmd == '0':
        esp.write(b'0')
        print("Fade dihentikan")
    elif cmd.lower() == 'q':
        esp.write(b'0')
        break

esp.close()
```

Penjelasan Kode Program

Pada Modul 3, program Python digunakan sebagai antarmuka kontrol untuk mengendalikan proses fade LED yang dijalankan oleh ESP32. Komunikasi antara Python dan ESP32 dilakukan melalui komunikasi serial menggunakan library serial.

Pada bagian awal program, library serial dan time diimpor untuk mendukung komunikasi serial dan pemberian jeda waktu. Program kemudian membuka koneksi serial dengan menentukan port komunikasi dan baud rate sebesar 115200. Jeda waktu diberikan setelah koneksi dibuka untuk memastikan ESP32 telah siap menerima data.

Program menampilkan petunjuk penggunaan kepada pengguna, yaitu perintah '1' untuk memulai proses fade LED, perintah '0' untuk menghentikan proses fade dan mematikan LED, serta perintah 'q' untuk keluar dari program.

Selanjutnya, program Python berjalan dalam sebuah perulangan tanpa batas yang menunggu input dari pengguna. Ketika pengguna memasukkan perintah '1', Python mengirimkan karakter '1' ke ESP32 melalui komunikasi serial. Perintah ini akan diterjemahkan oleh ESP32 sebagai sinyal untuk memulai proses fade LED. Ketika perintah '0' dimasukkan, Python mengirimkan karakter '0' yang berfungsi untuk menghentikan proses fade dan mematikan LED.

Apabila pengguna memasukkan perintah 'q', program Python akan mengirimkan perintah penghentian ke ESP32, kemudian keluar dari perulangan dan menutup koneksi serial secara aman. Penutupan koneksi ini bertujuan untuk mencegah terjadinya konflik port serial pada saat program dijalankan kembali.

Dengan demikian, program Python pada Modul 3 berperan sebagai pengendali sederhana berbasis teks yang memungkinkan pengguna untuk mengaktifkan dan menonaktifkan efek fade LED pada ESP32 secara interaktif melalui komunikasi serial.

Percobaan 4 Mengatur kecepatan Motor menggunakan slider

- Arduino

```
#include <WiFi.h>
#include <PubSubClient.h>

// ================ WIFI ================
const char* ssid = "SAPU";
const char* password = "korek111";

// ================ MQTT ================
const char* mqttServer = "broker.emqx.io";
const int mqttPort = 1883;
const char* mqttTopic = "kontrolbas";
const char* mqttTopicSpeed = "kontrolbas/speed";
```

```
// ================ PIN ================
int ledPin = 2;          // LED onboard
int IN1 = 27;            // Driver IN1
int IN2 = 26;            // Driver IN2
int ENA = 12;            // Driver ENA (PWM)

// ================ PWM ================
const int pwmChannel = 0;
const int pwmFreq = 30000;
const int pwmResolution = 8;
int duty = 0;

// ================ MQTT CLIENT ==========
WiFiClient espClient;
PubSubClient client(espClient);

// ======================================

void connectWiFi() {
  Serial.print("Connecting WiFi ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi CONNECTED");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
}

void reconnectMQTT() {
  while (!client.connected()) {
    Serial.print("Connecting MQTT...");
    String clientID = "ESP32-" + String(random(0xffff), HEX);

    if (client.connect(clientID.c_str())) {
      Serial.println("CONNECTED");
      client.subscribe(mqttTopic);
      client.subscribe(mqttTopicSpeed);
      Serial.print("Subscribe: ");
      Serial.println(mqttTopic);
```

```
    } else {
      Serial.print("FAILED, rc=");
      Serial.print(client.state());
      delay(3000);
    }
  }
}

// ================ MOTOR ================
void motorON() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);

  // soft start
  for (int d = 0; d <= 200; d += 20) {
    ledcWrite(pwmChannel, d);
    delay(100);
  }

  digitalWrite(ledPin, HIGH);
  Serial.println("MOTOR ON");
}

void motorOFF() {
  for (int d = 200; d >= 0; d -= 20) {
    ledcWrite(pwmChannel, d);
    delay(50);
  }

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(ledPin, LOW);

  Serial.println("MOTOR OFF");
}

// ============== MQTT CALLBACK ===========
void callback(char* topic, byte* payload, unsigned int length) {
  String msg = "";
  for (int i = 0; i < length; i++) {
    msg += (char)payload[i];
  }

  Serial.print("Topic: ");
  Serial.print(topic);
```

```
  Serial.print(" | Payload: ");
  Serial.println(msg);

  // ===== ON / OFF =====
  if (String(topic) == "kontrolbas") {
    if (msg == "1" || msg == "ON") {
      motorON();
    }
    if (msg == "0" || msg == "OFF") {
      motorOFF();
    }
  }

  // ===== SPEED SLIDER =====
  if (String(topic) == "kontrolbas/speed") {
    duty = msg.toInt();
    duty = constrain(duty, 0, 255);
    ledcWrite(pwmChannel, duty);

    Serial.print("Speed set: ");
    Serial.println(duty);
  }
}

// ================= SETUP =================
void setup() {
  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);

  ledcSetup(pwmChannel, pwmFreq, pwmResolution);
  ledcAttachPin(ENA, pwmChannel);
  ledcWrite(pwmChannel, 0);

  connectWiFi();

  client.setServer(mqttServer, mqttPort);
  client.setCallback(callback);
}

// ================= LOOP =================
void loop() {
```

```
  if (!client.connected()) {
    reconnectMQTT();
  }
  client.loop();
}
```

Penjelasan Kode Program

Pada Modul 4, sistem dikembangkan untuk mengendalikan motor DC serta mengatur kecepatan putaran motor secara jarak jauh menggunakan jaringan WiFi dan protokol MQTT. ESP32 berfungsi sebagai perangkat IoT yang terhubung ke jaringan internet dan menerima perintah kontrol dari broker MQTT.

Pada bagian awal program, digunakan library WiFi.h dan PubSubClient.h untuk mendukung koneksi WiFi serta komunikasi MQTT. Selanjutnya, didefinisikan parameter jaringan WiFi berupa SSID dan password, serta alamat broker MQTT yang digunakan sebagai perantara komunikasi data. Dua topik MQTT digunakan, yaitu satu topik untuk perintah ON dan OFF motor, serta satu topik tambahan untuk pengaturan kecepatan motor.

Program kemudian mendefinisikan pin-pin yang digunakan, meliputi pin LED sebagai indikator status sistem, dua pin arah motor yang terhubung ke driver motor, dan satu pin enable motor yang digunakan sebagai keluaran PWM. Selain itu, parameter PWM ditentukan dengan frekuensi sebesar 30 kHz dan resolusi 8 bit, sehingga nilai PWM berada pada rentang 0 hingga 255.

Fungsi connectWiFi() bertugas untuk menghubungkan ESP32 ke jaringan WiFi hingga koneksi berhasil. Setelah terhubung, alamat IP ESP32 ditampilkan pada serial monitor sebagai indikator keberhasilan koneksi. Fungsi reconnectMQTT() digunakan untuk menjaga kestabilan koneksi MQTT. Apabila koneksi terputus, ESP32 akan mencoba terhubung kembali dan melakukan subscribe ke topik MQTT yang telah ditentukan.

Pengendalian motor dilakukan melalui dua fungsi utama, yaitu motorON() dan motorOFF(). Pada fungsi motorON(), arah putaran motor diatur melalui driver motor, kemudian motor dijalankan secara bertahap menggunakan metode soft start dengan menaikkan nilai PWM secara perlahan untuk mengurangi lonjakan arus awal. LED indikator dinyalakan sebagai tanda bahwa motor sedang aktif. Sebaliknya, pada fungsi motorOFF(), nilai PWM diturunkan secara bertahap hingga nol untuk menghentikan motor secara halus, kemudian seluruh pin motor dimatikan dan LED indikator dipadamkan.

Fungsi callback() berfungsi untuk memproses pesan yang diterima dari broker MQTT. Pesan yang diterima dikonversi menjadi bentuk string dan diproses berdasarkan topik MQTT yang digunakan. Jika pesan diterima melalui topik kontrol ON dan OFF, maka sistem akan menyalakan atau mematikan motor sesuai perintah. Jika pesan diterima melalui topik pengaturan kecepatan, maka nilai yang diterima akan diubah menjadi nilai PWM dan digunakan untuk mengatur kecepatan motor secara real-time.

Pada fungsi setup(), seluruh pin diatur sebagai output, konfigurasi PWM dilakukan, serta koneksi WiFi dan MQTT diinisialisasi. Sementara itu, pada fungsi loop(), ESP32 secara terus-menerus memeriksa status koneksi MQTT dan menjalankan fungsi pemrosesan pesan agar sistem dapat merespons perintah dengan cepat.

Dengan demikian, Modul 4 menunjukkan penerapan konsep Internet of Things (IoT) pada pengendalian motor DC, di mana sistem dapat dikontrol dan dimonitor dari jarak jauh melalui jaringan internet menggunakan protokol MQTT.

Kesimpulan dan Analisis

Berdasarkan hasil perancangan, implementasi, dan pengujian pada keempat modul yang telah dilakukan, dapat disimpulkan bahwa sistem pengendalian LED dan motor DC menggunakan ESP32 dapat bekerja dengan baik melalui berbagai metode komunikasi dan kontrol, mulai dari komunikasi serial hingga berbasis jaringan internet.

Pada Modul 1, sistem berhasil mengimplementasikan pengendalian ON dan OFF LED serta motor DC menggunakan komunikasi serial antara Python dan ESP32. Modul ini membuktikan bahwa komunikasi serial dapat digunakan sebagai metode dasar dalam pengiriman perintah sederhana dari perangkat lunak ke perangkat keras. Modul ini menjadi fondasi penting dalam memahami cara ESP32 menerima dan mengeksekusi perintah dari luar sistem.

Pada Modul 2, sistem dikembangkan dengan memanfaatkan jaringan WiFi dan protokol MQTT untuk mengendalikan motor DC secara nirkabel. Hasil pengujian menunjukkan bahwa ESP32 mampu menerima perintah ON dan OFF dari broker MQTT dengan respons yang stabil. Penggunaan protokol MQTT memberikan fleksibilitas dalam pengendalian jarak jauh serta menunjukkan penerapan awal konsep Internet of Things (IoT) pada sistem kendali motor.

Pada Modul 3, sistem difokuskan pada pengaturan intensitas cahaya LED menggunakan metode Pulse Width Modulation (PWM). Efek fade dari terang ke redup berhasil dihasilkan secara halus dan dapat dikendalikan melalui perintah dari Python. Modul ini menunjukkan bahwa PWM merupakan metode yang efektif dalam mengatur tingkat kecerahan LED, serta memperlihatkan integrasi antara kontrol perangkat lunak dan sinyal keluaran perangkat keras.

Berdasarkan hasil pengujian pada Modul 4, ditemukan bahwa ketika nilai slider kecepatan berada di sekitar nilai PWM 120, motor DC tidak berputar meskipun sistem telah menerima perintah pengaturan kecepatan. Kondisi ini disebabkan oleh beberapa faktor teknis yang berkaitan dengan karakteristik motor dan sistem kendali PWM yang digunakan.

Motor DC memiliki tegangan dan torsi awal minimum (starting torque) yang harus terpenuhi agar motor dapat mulai berputar. Pada nilai PWM sekitar 120, tegangan efektif yang diterima oleh motor belum cukup untuk mengatasi hambatan awal seperti gesekan mekanik, beban motor, serta resistansi internal motor itu sendiri. Akibatnya, motor berada dalam kondisi aktif secara logika, namun tidak mampu menghasilkan putaran.

Selain itu, penggunaan metode PWM menyebabkan tegangan rata-rata yang diterima motor bergantung pada duty cycle. Pada nilai PWM yang rendah hingga menengah, pulsa sinyal yang diberikan masih terlalu kecil sehingga motor tidak memperoleh energi yang cukup untuk mulai bergerak. Hal ini umum terjadi pada motor DC tanpa sistem penguatan torsi awal.

Faktor lain yang memengaruhi kondisi ini adalah driver motor yang digunakan. Setiap driver motor memiliki batas efisiensi tertentu, di mana pada duty cycle rendah terjadi penurunan daya akibat losses internal driver. Kondisi ini semakin memperkecil energi yang diterima oleh motor.

Untuk mengatasi permasalahan tersebut, diterapkan solusi berupa pemberian nilai PWM minimum (minimum duty cycle) agar motor selalu menerima daya yang cukup untuk mulai berputar. Selain itu, metode soft start tetap dipertahankan untuk mencegah lonjakan arus saat motor dinyalakan.

Sebagai alternatif, sistem dapat dimodifikasi dengan memetakan nilai slider sehingga nilai PWM yang dikirimkan tidak berada di bawah batas minimum tertentu. Dengan cara ini, setiap perubahan slider akan tetap menghasilkan pergerakan motor yang stabil.

Dengan demikian, tidak berputarnya motor pada nilai PWM sekitar 120 merupakan kondisi yang wajar secara teknis dan disebabkan oleh keterbatasan torsi awal motor serta karakteristik sistem PWM dan driver motor. Penerapan batas minimum PWM atau teknik soft start menjadi solusi efektif untuk meningkatkan keandalan sistem kendali kecepatan motor.

Secara keseluruhan, keempat modul menunjukkan bahwa ESP32 merupakan platform yang andal dan fleksibel untuk pengendalian perangkat elektronik baik secara lokal maupun jarak jauh. Sistem yang dikembangkan memiliki potensi untuk diterapkan pada berbagai aplikasi, seperti otomasi rumah, sistem kendali industri sederhana, dan perangkat IoT. Untuk pengembangan selanjutnya, sistem dapat ditingkatkan dengan menambahkan fitur keamanan komunikasi, antarmuka grafis pengguna, serta sistem monitoring berbasis sensor agar kinerja dan fungsionalitas sistem menjadi lebih optimal.

Lampiran Foto Hasil Percobaan